Modélisation et calcul scientifique

Jocelyne Erhel

INSA, 2014-2015

Chapitre: algèbre linéaire sur matrices pleines

- opérations de base : BLAS
- exemple : multiplication de matrices
- opérations avancées : LAPACK
- exemple : factorisation de Cholesky
- version parallèle de BLAS : PBLAS
- exemples BLAS1 et BLAS2

Opérations de base : BLAS

Basic Algebraic Linear Subroutines (BLAS) : trois niveaux

- BLAS 1 : opérations entre vecteurs
- BLAS 2 : opérations entre matrices et vecteurs
- BLAS 3 : opérations entre matrices

Avantages

- portabilité
- performance : version constructeur
- lisibilité

http://www.netlib.org/blas/index.html

BLAS 1

Opérations entre vecteurs de longueur n
Nombre d'opérations en $O(n)$ et nombre de données en $O(n)$
Exemples : produit scalaire $s = x^T y$, SAXPY $y = y + a * x$

Opérations matrice-vecteur
Nombre d'opérations en $O(n^2)$ et nombre de données en $O(n^2)$

- produit matrice-vecteur $y = \alpha Ax + \beta y$
- calcul sur matrices triangulaires
- résolution de systèmes triangulaires
- correction de rang 1 (produit extérieur) $A = A + \alpha uv^T$
- correction symétrique de rang 2 $A = A + \alpha(uv^T + vu^T)$

BLAS 3

Opérations matrice-matrice
Nombre d'opérations en $O(n^3)$ et nombre de données en $O(n^2)$
Performance maximale

- produit de matrices $C = A * B + \beta C$
- multiplications de matrices triangulaires
- résolution de systèmes triangulaires à plusieurs seconds membres

Multiplication de matrices

Multiplication : $C = C + A * B$

- `A matrice n1 x n2`
- `B matrice n2 x n3`
- `C matrice n1 x n3`

Algorithm 1 Produit de matrices *ijk*

```
1: for i = 1, n1 do
2:     for j = 1, n3 do
3:         for k = 1, n2 do
4:             C(i, j) = C(i, j) + A(i, k) * B(k, j)
5:         end for
6:     end for
7: end for
```

$n1 * n2 * n3$ opérations

6 multiplications de matrices

Trois boucles commutables $\Rightarrow$ 6 écritures

1 boucle interne : BLAS 1

(i,j,k) ou (j,i,k) : produit scalaire ligne-colonne

(i,k,j) ou (k,i,j): saxpy sur des lignes

(j,k,i) ou (k,j,i) : saxpy sur des colonnes

2 boucles internes : BLAS 2

(i,j,k) ou (i,k,j) : produit ligne-matrice

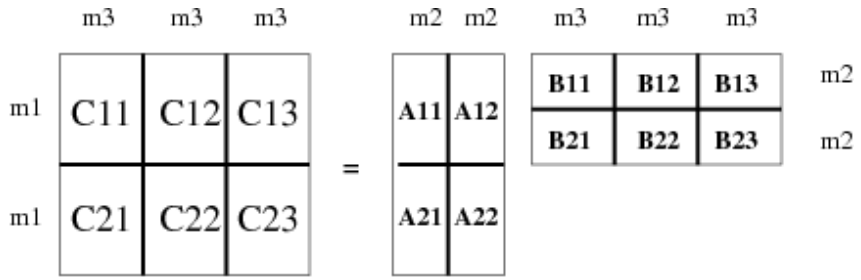
(j,i,k) ou (j,k,i): produit matrice-colonne

(k,i,j) ou (k,j,i) : correction de rang 1

Analyse des lectures

cas $n_1 = n_2 = n_3$ - version (i,j,k)
si B dans le cache, $O(n^2)$ lectures
sinon $O(n^3)$ lectures
Version par blocs pour utiliser le cache

Multiplication par blocs



n1 = k1 x m1, n2 = k2 x m2, n3 = k3 x m3
version (ikj)

Algorithm 2 Produit de matrices *ikj* par blocs

```
1: for i = 1, k1 do
2:     for k = 1, k2 do
3:         for j = 1, k3 do
4:             Cij = Cij + Aik x Bkj;
5:         end for
6:     end for
7: end for
```

Analyse des lectures dans la version par blocs

un bloc A_{ik} dans la mémoire cache
A_{ik} lu une fois - A lu une fois - $n_1 n_2$ mots
B_{kj} lu k_1 fois - B lu k_1 fois - $k_1 n_2 n_3$ mots
C_{ij} lu k_2 fois - C lu k_2 fois - $k_2 n_1 n_3$ mots
Total : $L = n_1 n_2 + k_1 n_2 n_3 + k_2 n_1 n_3 = n_1 n_2 + n_1 n_2 n_3 \left(\frac{1}{m_1} + \frac{1}{m_2} \right)$.

Taille optimale des blocs

Mémoire cache de $M + 1 bloc$ mots : $m_1 m_2 \leq M$
Stratégie optimale : $m_1 m_2 = min(M, n_1 n_2)$

Si $n_1 n_2 \leq M$: alors $m_1 = n_1$ et $m_2 = n_2$
sinon si $n_1 \leq \sqrt{M}$ et $n_2 > \sqrt{M}$: alors $m_1 = n_1$ et $m_2 = \frac{M}{m_1}$
sinon si $n_2 \leq \sqrt{M}$ et $n_1 > \sqrt{M}$: alors $m_2 = n_2$ et $m_1 = \frac{M}{m_2}$
sinon : $m_1 = m_2 = \sqrt{M}$

version BLAS optimisée pour chaque architecture

LAPACK

Résolution de problèmes d'algèbre linéaire

- Systèmes linéaires
- Problèmes linéaires aux moindres carrés
- Problèmes aux valeurs propres
- Problèmes aux valeurs singulières
- Estimation de conditionnement

Sur des matrices pleines ou bandes
Versions par blocs
BLAS3 le plus souvent possible
sinon BLAS2 et BLAS1
http://www.netlib.org/lapack/

Factorisation de Cholesky

$A \in \mathbb{R}^{n \times n}$ matrice symétrique $A^T = A$ définie positive $\forall x \neq 0,\ x^T A x > 0$
Factorisation de Cholesky $A = LL^T$ et $diag(L) > 0$
version récursive

$$A = \begin{pmatrix} \alpha & a^T \\ a & A_1 \end{pmatrix}$$

$$\begin{cases} \lambda = \sqrt{\alpha} \\ l = (1/\lambda)a \\ L_1 L_1^T = A_1 - l\, l^T \end{cases}$$

3 niveaux : 6 manières de dérouler la récursion

Cholesky : Right Looking ou Fan-out ou jki

Algorithm 3 Cholesky (Right Looking)

1: $L :=$ triangle_inférieur(A)
2: **for** $k = 1, n$ **do**
3: $\quad L(k,k) = \sqrt{L(k,k)}$
4: $\quad$ **for** $i = k+1, n$ **do**
5: $\quad\quad L(i,k) := L(i,k)/L(k,k)$
6: $\quad$ **end for**
7: $\quad$ **for** $j = k+1, n$ **do**
8: $\quad\quad$ **for** $i = j, n$ **do**
9: $\quad\quad\quad L(i,j) := L(i,j) - L(i,k) * L(j,k)$
10: $\quad\quad$ **end for**
11: $\quad$ **end for**
12: **end for**

BLAS2 et BLAS1

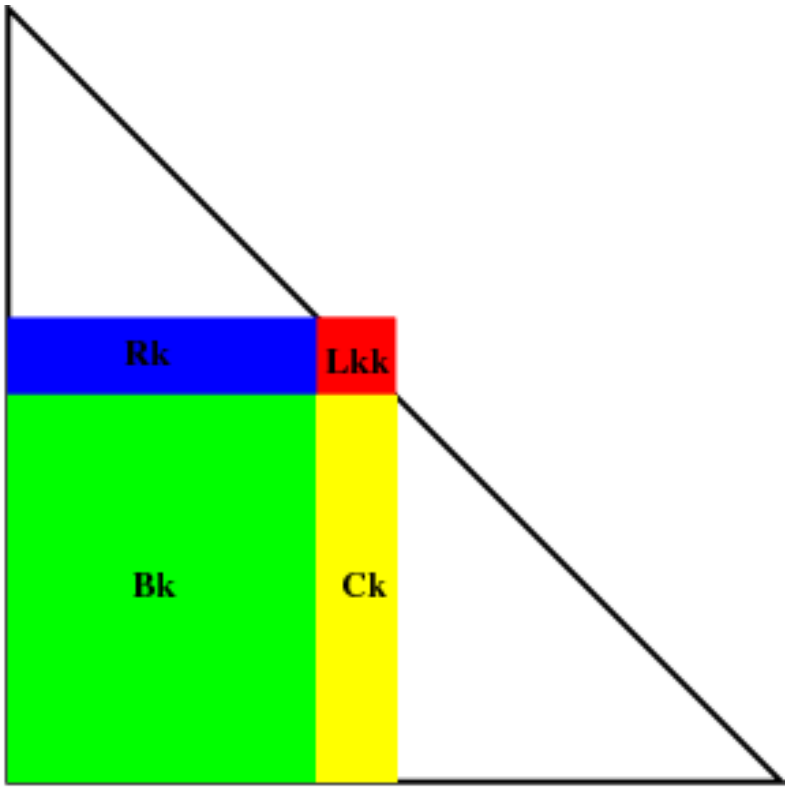
Cholesky : Left Looking ou Fan-in ou kji

Algorithm 4 Cholesky (Left Looking)

1: $L :=$ triangle_inférieur(A)
2: **for** $j = 1, n$ **do**
3: **for** $k = 1, j-1$ **do**
4: **for** $i = j, n$ **do**
5: $L(i,j) = L(i,j) - L(i,k) * L(j,k)$
6: **end for**
7: **end for**
8: $L(j,j) = \sqrt{L(j,j)}$
9: **for** $i = j+1, n$ **do**
10: $L(i,j) = L(i,j)/L(j,j)$
11: **end for**
12: **end for**

BLAS2 et BLAS1

Cholesky par blocs - Fan-in



Algorithm 5 Cholesky par blocs

1: $L = \text{triangle\_inférieur}(A)$
2: **for** $k = 1, nblocs$ **do**
3: $\quad L_{kk} = L_{kk} - R_k * R_k^T$
4: $\quad L_{kk} = chol(L_{kk})$
5: $\quad C_k = C_k - B_k * R_k^T$
6: $\quad C_k = C_k * L_{kk}^{-T}$
7: **end for**

BLAS3 et BLAS2

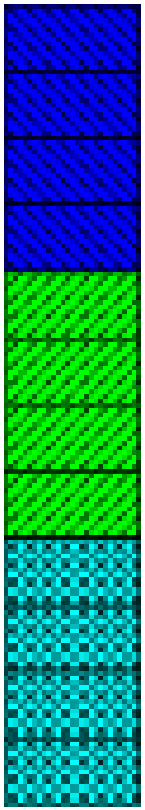
Parallel BLAS

Mémoire partagée
PLASMA: version parallèle de BLAS
Mémoire distribuée
PBLAS : Version distribuée de BLAS avec distribution des vecteurs et des matrices
communications par la bibliothèque BLACS

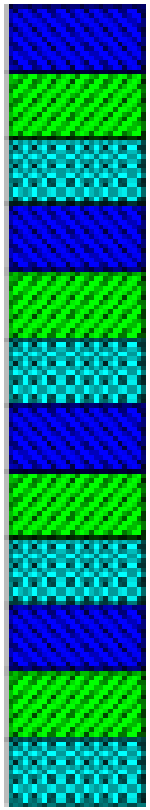
Distribution de vecteurs

P processus - Vecteur de longueur n

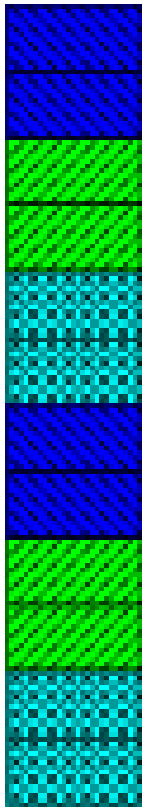
par blocs



cyclique



bloc-cyclique



paramètre d'optimisation : taille des blocs r

Opérations BLAS1

Algorithme du processus p=1,...P

Opération SAXPY

données locales : vecteurs x et y avec indices $j \in I_p$, scalaire a

Algorithm 6 SAXPY:$y = y + a * x$

1: **for** $j \in I_p$ **do**
2: $\quad y(j) = y(j) + a * x(j)$
3: **end for**

Opération DOT

données locales : vecteurs x et y avec indices $j \in I_p$, scalaires a_p et a

Algorithm 7 DOT:$a = \sum_{j=1}^{n} x_j * y_j$

1: $a_p = 0$
2: **for** $j \in I_p$ **do**
3: $\quad a_p = a_p + x(j) * y(j)$
4: **end for**
5: $a = GLOBALSUM(a_p)$

Algorithme du processus p=1,...P

Opération y=y+A*x

données locales : vecteurs x et y et y_p avec indices $j \in I_p$

matrice A avec colonnes $j \in I_p$

Algorithm 8 MATVEC

1: $y = 0$
2: **for** $j \in I_p$ **do**
3: **for** $i = 1, n$ **do**
4: $y_p(i) = y_p(i) + a(i,j) * x(j)$
5: **end for**
6: **end for**
7: **for** $j \in I_p$ **do**
8: $y(j) = GLOBALSUM(y_p(j))$
9: **end for**